Attorney Docket No. SUN1P816/P5614

PATENT APPLICATION

LOADING ATTRIBUTE FOR PARTIAL LOADING OF CLASS FILES INTO VIRTUAL MACHINES

Inventors: 1. Stepan Sokolov
34832 Dorado Common
Fremont, CA 94555
Citizenship: Ukraine

2. David Wallman
777 S. Mathilda Ave., #266
Sunnyvale, CA 94087
Citizenship: Israel

Assignee: Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303

BEYER WEAVER & THOMAS, LLP
P.O. Box 778
Berkeley, CA 94704-0778
Telephone (650) 961-8300

LOADING ATTRIBUTE FOR PARTIAL LOADING OF CLASS FILES INTO VIRTUAL MACHINES

CROSS-REFERENCE TO RELATED APPLICATIONS

This application is related to U.S. Patent Application No. 09/703,361 (Att.Dkt.No. SUN1P809/P5500), entitled "IMPROVED FRAMEWORKS FOR INVOKING METHODS IN VIRTUAL MACHINES," which is hereby incorporated herein by reference.

This application is related to U.S. Patent Application No. 09/703,356 (Att.Dkt.No. SUN1P810/P5510), entitled "IMPROVED METHODS AND APPARATUS FOR NUMERIC CONSTANT VALUE INLINING IN VIRTUAL MACHINES," which is hereby incorporated herein by reference.

This application is related to U.S. Patent Application No. 09/703,449 (Att.Dkt.No. SUN1P814/P5417), entitled "IMPROVED FRAMEWORKS FOR LOADING AND EXECUTION OF OBJECT-BASED PROGRAMS," which is hereby incorporated herein by reference.

This application is related to U.S. Patent Application No. _____ (Att.Dkt.No. SUN1P815/P5612), entitled "TECHNIQUES FOR LOADING CLASS FILES INTO VIRTUAL MACHINES," which is hereby incorporated herein by reference.

BACKGROUND OF THE INVENTION

1. The Field of the Invention

The present invention relates generally to object-oriented programming environments. More specifically, the invention relates to improved frameworks for loading class files into virtual computing machines.

2. The Relevant Art

The present invention relates generally to object-based high level programming environments, and more particularly, to virtual machine instruction sets suitable for execution in virtual machines operating in portable, platform independent programming environments.

Recently, the Java™ programming environment has become quite popular. The Java™ programming language is a language that is designed to be portable enough to be executed on a wide range of computers ranging from small devices (e.g., pagers, cell phones and smart cards) up to supercomputers. Computer programs written in the Java programming language (and other languages) may be compiled into Java Bytecode instructions that are suitable for execution by a Java virtual machine implementation.

The Java virtual machine is commonly implemented in software by means of an interpreter for the Java virtual machine instruction set but, in general, may be software, hardware, or both. A particular Java virtual machine implementation and corresponding support libraries together constitute a Java™ runtime environment.

Computer programs in the Java programming language are arranged in one or more classes or interfaces (referred to herein jointly as classes or class files). Such programs are generally platform, i.e., hardware and operating system, independent. As such, these computer programs may be executed, without modification, on any computer that is able to run an implementation of the Java™ runtime environment.

Object-oriented classes written in the Java programming language are compiled to a particular binary format called the "class file format." The class file includes various components associated with a single class. These components can be, for example, methods and/or interfaces associated with the class. In addition, the class file format includes a significant amount of ancillary information that is associated with the class. Fig. 1A illustrates a class file structure. The class file format (as well as the general operation of the Java virtual machine) is described in some detail in The Java Virtual

Cited IDS

Machine Specification by Tim Lindholm and Frank Yellin (ISBN 0-201-31006-6), which is hereby incorporated herein by reference.

Generally, when a class file is loaded into the virtual machine, the virtual machine essentially makes a copy of the class file for its internal use. The virtual machine's internal copy is sometimes referred to as an "internal class representation." In conventional virtual machines, the internal class representation is typically almost an exact copy of the class file. This is true regardless of whether the loaded information is likely to be used or not. For example, typically, an exact copy of common Java classes (e.g., class PrintWriter) are loaded into the virtual machine. These common classes typically have a large size. For example, the class PrintWriter may take up as much as 40 KiloBytes (40 K) of memory. However, 90% of the common class is often not used during the execution of a computer program. This, of course, results in a grossly inefficient use of memory resources. In some circumstances, particularly in embedded systems with limited memory resources, this inefficient use of memory resources is a significant disadvantage.

In view of the foregoing, improved techniques for loading class files into virtual computing machines are needed.

SUMMARY OF THE INVENTION

To achieve the foregoing and other objects of the invention, improved techniques for loading class files into virtual computing machines are disclosed. One aspect of the present invention seeks to provide a mechanism that will generally improve the efficiency of virtual machines by selectively loading information into a virtual machine. In other words, unlike conventional techniques where the entire class file is substantially loaded into the virtual machine, the inventive techniques can operate to load only a portion of the class file. As will be appreciated, this allows a better use of the resources of the virtual machine. The inventive mechanisms are especially

effective in virtual machines that operate with limited memory resources (e.g., embedded systems).

In accordance with the Java virtual machine specification, new class attributes can be defined for, and used by, Java virtual machine implementations. In one embodiment, a new class attribute ("load-attribute") is defined and implemented for class files. This can be, for example, implemented as a "load-attribute" table that lists components that have been selected for loading into the virtual machine. In addition, the load-attribute may provide references to the selected components in the class file. Accordingly, various components in the class file can be marked for loading and selectively loaded.

The invention can be implemented in numerous ways, including as a method, an apparatus, a computer readable medium, and a database system. Several embodiments of the invention are discussed below.

As a method for loading a class file into a virtual machine, one embodiment of the invention includes that acts of: determining whether one or more components of the class have been marked to be loaded into the virtual machine; and loading said one or more components of the class into the virtual machine when one or more components of the class file have been marked to be loaded into said virtual machine.

As a class file suitable for loading into a virtual machine, one embodiment of the invention includes a load attribute portion that provides information about one or more components of said class which have been marked to be loaded into said virtual machine.

As a computer readable medium including computer readable code for representing a class file suitable for loading into a virtual machine, one embodiment of the invention includes computer readable code representing a load attribute portion for said class file. The attribute portion lists one or more components of said class which have been marked to be loaded into said virtual machine.

These and other aspects and advantages of the present invention will become more apparent when the detailed description below is read in conjunction with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

The present invention will be readily understood by the following detailed description in conjunction with the accompanying drawings, wherein like reference numerals designate like structural elements, and in which:

Fig. 1A illustrates a class file structure.

Fig. 2A is a representation of an object-oriented computing environment including a class file, and an internal class representation of the class file inside a virtual machine in accordance with one embodiment of the invention.

Fig. 2B represents an object-oriented computing environment including a raw class heap in accordance with one embodiment of the invention.

Fig. 3A illustrates a class file with one or more components marked in accordance with one embodiment of the invention.

Fig. 3B illustrates a class attribute portion of a class file in accordance with one embodiment of the invention.

Fig. 4 illustrates a loading method for loading a class file into the virtual machine in accordance with one embodiment of the invention.

Fig. 5 illustrates in greater detail a loading method for loading a class file into the virtual machine in accordance with another embodiment of the invention.

DETAILED DESCRIPTION OF THE INVENTION

As described in the background section, the Java programming environment has enjoyed widespread success. Therefore, there are continuing efforts to extend the breadth of Java compatible devices and to improve the performance of such devices. One of the most significant factors influencing the performance of Java based programs on a particular platform is the performance of the underlying virtual machine. Accordingly, there have been extensive efforts by a number of entities to provide improved performance to Java compliant virtual machines. In order to be Java compliant, a virtual machine must be capable of working with Java classes which have a defined class file format. Although it is important that any Java virtual machine be capable of handling Java classes, the Java virtual machine specification does not dictate how such classes are represented internally within a particular Java virtual machine implementation. In addition, in accordance with the Java virtual machine specification, new class attributes can be defined for, and used by, Java virtual machine implementations.

The present invention pertains to improved frameworks for loading class files into virtual computing machines. One aspect of the present invention seeks to provide a mechanism that will generally improve the efficiency of virtual machines by selectively loading information into a virtual machine. In other words, unlike conventional techniques where the entire class file is substantially loaded into the virtual machine, the inventive techniques can operate to load only a selected portion of the class file. As will be appreciated, this allows for better use of the resources. Thus, the inventive mechanisms are especially effective in virtual machines that operate with limited memory resources (e.g., embedded systems).

In one embodiment, a new class attribute ("load-attribute") is defined and implemented for class files. This can be, for example, implemented as a "load-attribute" table that lists the components that have been selected for loading into the virtual machine. In addition, the load-attribute may provide references to the selected components in the class file. Accordingly, various

components of the class file can be marked for loading and then selectively loaded.

Embodiments of the invention are discussed below with reference to Figs. 2-5. However, those skilled in the art will readily appreciate that the detailed description given herein with respect to these figures is for explanatory purposes as the invention extends beyond these limited embodiments.

Fig. 2A is a representation of an object-oriented computing environment 200 including a class file 202, and an internal class representation 204 of the class file 202 inside a virtual machine 206 in accordance with one embodiment of the invention. The class file 202 typically resides in a secondary storage device outside the virtual machine 206. However, the class file 202 is represented as the internal class representation 204 inside the virtual machine 206.

As will be appreciated, various components of the class file 202, for example, components C_i and C_j of the class file 202 can be marked to indicate that these selected components are to be loaded into the virtual machine 206. The selected components C_i and C_j can, for example, represent the components of the class file that will be required at run time. Accordingly, only the marked components C_i and C_j of the class file 202 need to be loaded into the virtual machine 206. Thus, rather than representing the entire class file 202 inside the virtual machine 206, the internal representation 204 consists of only marked components C_i and C_j. This allows for selective loading of class files and generally improves the efficiency of virtual machines, especially those with limited resources.

In addition, the present invention can be implemented in conjunction with a raw-class heap portion to further improve efficiency. Fig. 2B represents an object-oriented computing environment 250 including a raw class heap 252 in accordance with one embodiment of the invention. The raw-class heap portion 252 represents a portion of the memory of the object-oriented computing environment 250 that is used to initially load class files. This

memory can be, for example, a memory portion of the object-oriented computing environment 252 that is dedicated to loading class files (i.e., dedicated memory). More details about the implementation of a raw class heap are disclosed in the related to U.S. Patent Application No. _____ (Att.Dkt.No. SUN1P815/P5612), entitled "TECHNIQUES FOR LOADING CLASS FILES INTO VIRTUAL MACHINES".

As noted above, various components of the class file 202 can be marked for loading into the virtual machine. Fig. 3A illustrates a class file 202 with one or more components marked in accordance with one embodiment of the invention. Again, these marked components, can be, for example, components C_i and C_j of the class file 202. In the described embodiment, the class file 202 includes a class attributes portion 302. The class attributes portion 302 can be implemented to include the information relating to the marked components C_i and C_j of the class file 202. As will be appreciated, this can be achieved by defining a new class attribute since Java virtual machine implementations are allowed to define new attributes for class files. The new attribute can be defined in the "attributes" portion (table) of the class file structure shown in Fig. 1.

Accordingly, the class attributes portion 302 can be implemented in the attributes table of the class file 202 as a new attribute, for example, as a "load-attribute". As will be appreciated, this approach offers many advantages since Java virtual machines are permitted to recognize and use newly defined attributes found in the attributes tables of the class file structure. Thus, the invention can be implemented without requiring a tremendous amount of change to the existing virtual machine implementations. Furthermore, Java virtual machines are required to ignore the attributes that are not recognized. Thus, defining the "load-attribute" in the attributes portion of the class file 202 still allows the loading of the class file 202 in conventional virtual machines which do not recognize an attribute for loading of the components.

Fig. 3B illustrates a class attributes portion 302 of a class file 202 in accordance with one embodiment of the invention. In the described

embodiment, the class attributes portion 302 is implemented as a "load-attribute" table 302. The load-attribute table 302 can include one or more entries corresponding to the marked components of the class file 202. For example, the first entry in the load-attribute table 302 is an offset C_i 304 representing the offset of the component C_i 306 in the class file 202. Similarly, the second entry in the load-attribute table 302 is an offset C_j 308 that represents the offset of the component C_j 310 in the class file 202. In this manner, the load-attribute table 302 can provide information about the components that are marked for loading.

Accordingly, the load-attribute table 302 can be used to determine which components of a class file should be loaded into the virtual machine. Fig. 4 illustrates a loading method 400 for loading a class file into the virtual machine in accordance with one embodiment of the invention. The loading method 400 is suited for selectively loading various components of class files into the virtual machine. Initially, at operation 402, a determination is made as to whether one or more components of the class file have been marked for loading into the virtual machine. If it is determined at operation 402 that no component of the class file has been marked, the loading method 400 proceeds to operation 404 where the class file is not loaded into the virtual machine. In other words, the class file is effectively ignored, however, this means that the class file may be loaded later if the need arises. Thereafter, the loading method 400 ends. However, if it is determined at operation 402 that at least one component of the class file has been marked, the loading method 400 proceeds to operation 406 where the one or more components marked for loading are loaded into the virtual machine.

To further elaborate, Fig. 5 illustrates in greater detail a loading method 500 for loading a class file into the virtual machine in accordance with another embodiment of the invention. At operation 502, sequential reading of the class file is initiated. The sequential reading of the class file is initiated from the beginning of the class file. Next, at operation 504, a determination is made as to whether a "load-attribute" table has been found. As noted above, the "load-attribute" table includes the information regarding the components

of the class that have been marked to be loaded. If it is determined at operation 504 that the load-attribute" table has not been found, the loading method 500 proceeds to operation 506 where a determination is made as to whether the end of the class file has been reached. If it is determined at operation 506 that the end of the class file has been reached, the loading method 500 ends. However, if it is determined at operation 506 that the end of the class file has not been reached, the loading method 500 proceeds to operation 508 where the sequential read of the class file is continued. The loading method 500 then proceeds to operation 504 where a determination is made as to whether a "load-attribute" table has been found.

If it is determined at operation 504 that the "load-attribute" table has been found, the loading method 500 proceeds to operation 510 where the "load-attribute" table is read. The loading method 500 then proceeds to operation 512 where a second sequential reading of the class file is initiated. Next, at operation 514, it is determined whether a component of the class file has been encountered. If it is determined at operation 514 that a component of the class file has not been encountered, the loading method 500 proceeds to operation 516, where a determination is made as to whether the end of the class file has been reached. If it is determined at operation 516 that the end of the class file has been reached, the loading method 500 ends. However, if it is determined at operation 516 that the end of the class file has not been reached, the loading method 500 proceeds to operation 518 where the second sequential reading of the class file is continued.

When it is determined at operation 514 that a component of the class file has been encountered, the loading method 500 proceeds to operation 520 where a determination is made as to whether the encountered component of the class is in the "load-attribute" table. If it is determined that the encountered component of the class is not in the "load-attribute" table, the loading method 500 proceeds to operation 516 where it is determined whether the end of the file has been reached. Thereafter, the loading method 500 proceeds in a similar manner as described above.

However, if it is determined at operation 520 that the encountered component is in the "load-attribute" table, the encountered component is read from the class file at operation 522. Next, at operation 524, an internal representation of the component is created inside the virtual machine. The creation of the internal representation can, for example, include the operations of creating a shell and populating it in accordance with the inventions described in U.S. Patent Application No. 09/703,361 (Att.Dkt.No. SUN1P809/P5500), entitled "IMPROVED FRAMEWORKS FOR INVOKING METHODS IN VIRTUAL MACHINES," U.S. Patent Application No. 09/703,356 (Att.Dkt.No. SUN1P810/P5510), entitled "IMPROVED METHODS AND APPARATUS FOR NUMERIC CONSTANT VALUE INLINING IN VIRTUAL MACHINES," and U.S. Patent Application No. 09/703,449 (Att.Dkt.No. SUN1P814/P5417), entitled "IMPROVED FRAMEWORKS FOR LOADING AND EXECUTION OF OBJECT-BASED PROGRAMS."

After the creation of the internal representation at operation 524, the loading method 500 proceeds to operation 516 where a determination is made as to whether the end of the file has been reached. Thereafter, the loading method 500 proceeds in a similar manner as described above. When it is determined at operation 516 that the end of the class file has been reached, the loading method 500 ends.

The many features and advantages of the present invention are apparent from the written description, and thus, it is intended by the appended claims to cover all such features and advantages of the invention. Further, since numerous modifications and changes will readily occur to those skilled in the art, it is not desired to limit the invention to the exact construction and operation as illustrated and described. Hence, all suitable modifications and equivalents may be resorted to as falling within the scope of the invention.

What is claimed is: